Programare multiparadigmă - **JAVA**

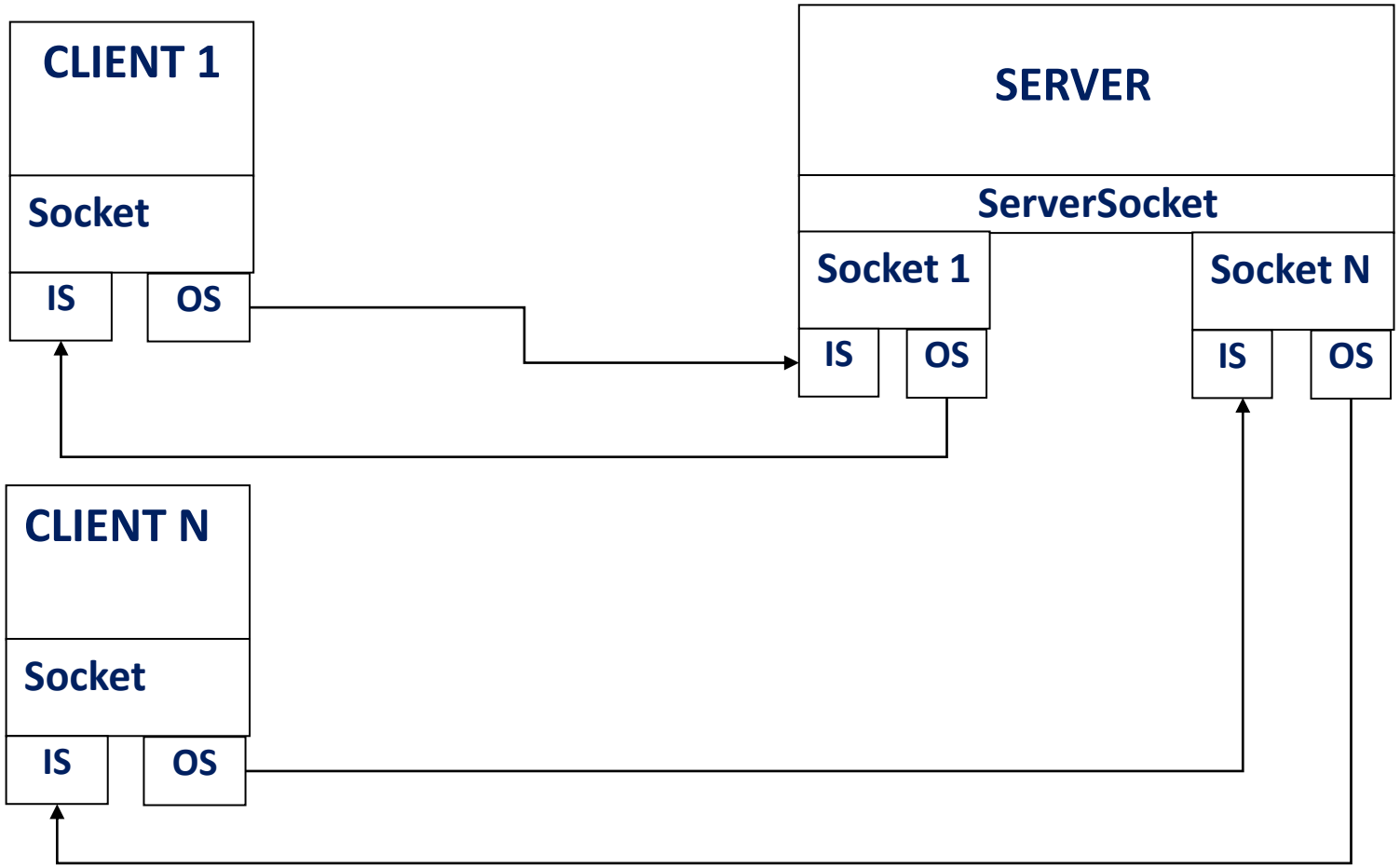
Prof. univ. dr. **Claudiu Vințe**

claudiu.vinte@ie.ase.ro

Comunicarea prin *TCP/IP*

- *Internet Protocol* – ***IP***:
  - Permite transmisia de pachete de date (*datagram*) între calculatoare
  - Calculatoarele sunt identificate printr-o adresă (32 biți – IPv4 sau 128 biți – IPv6)
- *Transmission Control Protocol* – ***TCP***:
  - Protocol bazat pe conexiuni care utilizează *IP* pentru transferul datelor
  - Poate fi utilizat simultan de mai multe procese identificate unic printr-un număr pe 16 biți denumit ***port***
  - Asigură:
    - Retransmisia pachetelor pierdute
    - Reordonarea pachetelor primite în altă ordine decât ordinea în care au fost transmise
    - Detecţia erorilor
    - Controlul vitezei de transfer
- O aplicație bazată pe TCP conține uzual:
  - O componentă server care răspunde la cererile inițiate de către clienți
  - Mai multe componente client care inițiază conexiunile

Conexiuni TCP - Arhitectura



Crearea unui server TCP/IP

- ***Etapa 1***: Construirea unui obiect de tip ***ServerSocket*** și specificarea portului prin intermediul constructorului
  - Numărul portului poate fi orice port liber (uzual >1024) sau 0 pentru un port alocat automat.
- ***Etapa 2***: Acceptarea unei conexiuni și obținerea obiectului ***Socket***
  - Se realizează prin apelul metodei ***accept;*** aceasta blochează firul de execuție curent până la primirea unei conexiuni sau expirarea timpului de așteptare setat prin intermediul metodei *setSoTimeout*
- ***Etapa 3***: Obținerea obiectelor *stream* din *Socket* și utilizarea acestora pentru comunicare
  - Datele transmise de către client sunt disponibile prin intermediul unui obiect de tip *InputStream* obținut prin apelul metodei *getInputStream*;
  - Datele sunt transmise prin scrierea în obiectul de tip *OutputStream* obținut prin metoda *getOutputStream*
- Resursele trebuie eliberate prin utilizarea *try-with-resources* sau apelarea metodei *Close* în interiorul unei clauze *finally*
- Excepția *IOException* trebuie declarată sau tratată
- Adresa clientului (de tip *InetAddress*) poate fi obținută prin apelul metodei *getInetAddress*.

Crearea unui server TCP/IP

Exemplu:

```
final int PORT_NUMBER = 8193;
try (ServerSocket serverSocket = new ServerSocket(PORT_NUMBER)) {
    while (true) {
        try (Socket socket = serverSocket.accept();
             BufferedReader in = new BufferedReader(
                     new InputStreamReader(socket.getInputStream()));
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        ) {
            String mesajPrimit = in.readLine();
            out.printf("Mesajul primit este: %s%n", mesajPrimit);
        }
    }
}

// Valoare true a celui de al doilea parametru al contructorului PrintWriter înseamnă că
// se șterg automat datele din buffer-ul de ieșire odată ce sunt apelate metode ce
// generează trecerea pe o nouă linie, cum ar fi println(), printf(), sau format()
// sau orice caracter newline ('\n') este găsit în șir.
// În acest fel nu trebuie apelată explicit metoda flush().
```

Crearea unui client TCP/IP

- *Etapa **1***: Construirea unui obiect de tip ***Socket*** și specificarea adresei și portului aplicației server prin intermediul constructorului
  - Constructorul inițiază conexiunea cu aplicația server
- *Etapa 2*: Obținerea obiectelor *stream* din *Socket* și utilizarea acestora pentru comunicare
  - Datele sunt transmise folosind obiecte *InputStream* și *OutputStream* obținute prin apelul metodelor *getInputStream* și *getOutputStream*

Exemplu:

```
final int PORT_NUMBER = 8193;
try (Socket socket = new Socket("127.0.0.1", PORT_NUMBER);
     BufferedReader in = new BufferedReader(
             new InputStreamReader(socket.getInputStream()));
     PrintWriter out = new PrintWriter(socket.getOutputStream(), true);) {

    out.println("Test");
    System.out.println(in.readLine());
}
```

Server TCP/IP multi-client

Este pornit câte un fir de execuție pentru fiecare cerere:

```
public static void main(String[] args) throws IOException {
    final int PORT_NUMBER = 8193;
    try (ServerSocket serverSocket = new ServerSocket(PORT_NUMBER)) {
        while (true) {
            Socket socket = serverSocket.accept();
            new Thread(() -> procesareCerere(socket)).start();
                  // Runnable este o interfață funcțională
        }
    }
}

private static void procesareCerere(Socket paramSocket) {
    try (Socket socket = paramSocket;
         BufferedReader in = new BufferedReader(
                 new InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    ) {
        // procesare cerere
    } catch (IOException exception) {
        exception.printStackTrace();
    }
}
```

Comunicarea prin UDP

- *User Datagram Protocol* – ***UDP***
  - un protocol ce trimite pachete de date independente, numite *datagrame*, de la un calculator către altul fără a garanta în vreun fel ajungerea acestora la destinație, ordinea sau eliminarea mesajelor duplicate
  - nu stabilește o conexiune permanentă între cele două calculatoare
- O *datagramă* conține:
  - Portul sursă – identifică portul utilizat de către procesul care a transmis mesajul utilizat de către aplicația server atunci când construiește obiectul răspuns
  - Portul destinație – identifică portul utilizat de către procesul care primește mesajul; utilizat de către sistemul de operare pentru transmiterea mesajului către procesul corect
  - Lungimea datelor în octeți
  - Conținutul mesajului (un vector de octeți)

Crearea unui server UDP

Crearea unui server UDP presupune parcurgerea următoarelor etape:

1. Crearea unui obiect de tip ***DatagramSocket*** cu *specificarea portului* utilizat
2. Crearea unui vector de octeți și a unui obiect ***DatagramPacket*** pentru recepționarea mesajului
3. Recepționarea mesajului prin apelul metodei ***receive***; apelul blochează firul de execuție până la primirea unui mesaj de la un client
4. Extragerea datelor, adresei și portului clientului din mesajul primit
5. Construirea obiectului ***DatagramPacket*** pentru răspuns pe baza adresei și portului specificate în cerere
6. Transmiterea răspunsului către aplicația client prin metoda ***send***

Crearea unui server UDP

Exemplu:

```
final int PORT_NUMBER = 1193;
final int MAX_MESSAGE_SIZE = 65535;

try (var socket = new DatagramSocket(PORT_NUMBER)) {
    while (true) {
        var buffer = new byte[MAX_MESSAGE_SIZE];
        var cerere = new DatagramPacket(buffer, buffer.length);
        socket.receive(cerere);

        // preluare date: cerere.getData()

        InetAddress adresaClient = cerere.getAddress();
        int portClient = cerere.getPort();
        // buffer = ...; // construire răspuns
        var raspuns = new DatagramPacket(buffer, buffer.length, adresaClient, portClient);
        socket.send(raspuns);
    }
}
```

Crearea unui client UDP

Crearea unui client UDP presupune parcurgerea următoarelor etape:

1. Crearea unui obiect de tip ***DatagramSocket*** *fără* port specificat
2. Crearea unui vector de octeți și a unui obiect ***DatagramPacket*** pentru mesajul cerere. Trebuie specificate adresa și portul serverului.
3. Transmiterea cererii către aplicația server prin metoda ***send***
4. Recepționarea răspunsului prin apelul metodei ***receive***; apelul blochează firul de execuție până la primirea răspunsului de la server.
5. Extragerea datelor din răspunsul primit de la server

Crearea unui client UDP

Exemplu:

```
final String SERVER_HOST_NAME = "localhost";
final int SERVER_PORT_NUMBER = 1193;
final int MAX_MESSAGE_SIZE = 65535;

try (var socket = new DatagramSocket()) {
    // 1. Transmitere cerere UDP
    var buffer = "cerere".getBytes();
    var cerere = new DatagramPacket(
            buffer,
            buffer.length,
            InetAddress.getByName(SERVER_HOST_NAME),
            SERVER_PORT_NUMBER);
    socket.send(cerere);

    // 2. Receptionare raspuns
    buffer = new byte[MAX_MESSAGE_SIZE];
    var raspuns = new DatagramPacket(buffer, buffer.length);
    socket.receive(raspuns);
    // utilizare răspuns obținut din raspuns.getBytes()
}
```

Utilizare URL

- URL este acronimul pentru *Uniform Resource Locator* și reprezintă adresa unei resurse aflată în Internet
- Un URL conține următoarele informații:
  - identificatorul de protocol (Ex. http)
  - identificator resursă (nume de host, port comunicare, cale resursă în cadrul serverului)
- Clasa care permite lucrul cu URL-uri este java.net.URL:
  - ***public final class URL extends Object implements Serializable;***
- Constructori:
  - *URL(String spec)* - Creează un URL pornind de la reprezentarea String a acestuia
    - exemplu: *URL urlServer = new URL("http://www.ase.ro");*
  - *URL(String protocol, String host, int port, String file)* - Creează un obiect URL specificând distinct toate elementele componente
- Există o serie de metode care furnizează informațiile conținute într-un URL: *getProtocol, getHost, getPort, getPath, getFile, ...*

Utilizare URL – Citirea datelor

- Citirea conținutului unui URL se realizează prin intermediul unui obiect *InputStream* furnizat prin metoda *openStream()* a clasei URL

Exemplu:

```
try {
    URL url = new URL("https://www.ase.ro");
    try (BufferedReader in = new BufferedReader(new
            InputStreamReader(url.openStream()))) {
        in.lines()
        .forEach(linie -> System.out.println(linie));
    }
} catch (Exception ex) {
    System.err.println(ex);
}
```

Utilizare URL – Transmiterea datelor

- Se utilizează metoda *openConnection()* a clasei *URL*, care furnizează un obiect ***URLConnection*** responsabil cu stabilirea unei conexiuni bidirecționale cu resursa specificată:
  - *public URLConnection openConnection() throws IOException;*
- Clasa abstractă *URLConnection* este superclasa tuturor claselor care reprezintă o legătură de comunicare între aplicație și o adresă URL (Exemplu: *HttpURLConnection*)
- Fluxurile de comunicare sunt furnizate de metodele clasei *URLConnection*:
  - *public InputStream* ***getInputStream****() throws IOException;*
  - *public OutputStream* ***getOutputStream****() throws IOException;*
- O conexiune URL poate fi utilizată pentru citire și/sau scriere. Controlul scrierii și citirii se realizează prin două câmpuri de tip boolean, doInput și doOutput, care pot fi modificate prin metodele:
  - *public void setDoInput(boolean doinput);*
  - *public void setDoOutput(boolean dooutput);*

Server HTTP

- *HyperText Transfer Protocol* - ***HTTP*** este un protocol de comunicație responsabil cu transferul de hipertext (text structurat ce conține legături) dintre un client (de regulă un navigator web) și un server web

- Clasa care permite crearea unui server HTTP simplu este ***HttpServer***
- Crearea unui obiect HttpServer se realizează prin metoda statică ***create()*** a clasei *HttpServer:*
    - *public static HttpServer create(InetSocketAddress addr, int backlog)* - sunt specificate adresa serverului și numărul maxim de conexiuni în așteptare

- Pornirea serverului se face prin invocarea metodei ***start()***

- Presupune crearea unui obiect ***HttpContext***.
- Un obiect *HttpContext* reprezintă o mapare dintre o cale ***URI*** (*Uniform Resource Identifier*) și o metodă care tratează cererile pe acest *HttpServer*.
    - *public abstract HttpContext* ***createContext****(String path, HttpHandler handler);*

Server HTTP

- Metoda *handler* este furnizată printr-un obiect care implementează interfața *HttpHandler*.
- Interfața *HttpHandler* are o singură metodă abstractă prin care se implementează codul metodei handler care tratează cererile:
  - *void handle(HttpExchange exchange) throws IOException;*

- Clasa *HttpExchange* încapsulează o cerere HTTP și un răspuns care urmează să fie generat. Aceasta oferă metode pentru examinarea cererii de la client și pentru construirea și trimiterea răspunsului.
- Pentru construirea unui server simplu care să comunice cu aplicații client prin conexiunile *URLConnection* ale clienților, vor fi utilizate metodele *HttpExchange* care furnizează fluxuri de acces la conținutul cererii și răspunsului
  - *public abstract InputStream getRequestBody();*
  - *public abstract OutputStream getResponseBody();*